

SW Engineering CSC 648 Summer 2021

DormMates

Milestone 2 • Version 2 • 08 July 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M2V2	7/13/2021	Updated database business rules, functional requirements, uml diagram, application and deployment diagram.
M2V1	7/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	6/22/2021	Initial submission

Table of Contents

Table of Contents	3
Data Definitions	4
Prioritized Functional Requirements	7
Priority 1 - Required	7
Priority 2 - Desired	8
Priority 3 - Opportunistic	9
UI Mockups and Storyboards	10
UI Mockups	10
Storyboards	10
High Level Database Architecture and Organization	24
Business Rules	24
Entities	25
ERD Diagram	28
Database Model	29
DBMS / Information	30
High Level APIs and Main Algorithms	31
High Level UML Diagrams	35
High Level Application Network and Deployment Diagrams	37
Network Diagram	37
Deployment Diagram	38
Key Risks	39
Skills	39
Schedule	39
Technical	39
Teamwork	40
Legal/Content	40
Project Management	41
Detailed List of Contributions	42

Data Definitions

- **General User:** A user who can view and access the website limited to the website's features. A general user must **register** by creating a student or landlord account to gain more privileges.
 - **Registration:** A general user has the option to create an account.
 - A general user shall be able to view limited listings based on an institution.
 - A general user shall be able to view the informational pages regarding the service.
- **Registered User:** A user who has access to the website's core features.
 - **Account Contains:** All registered user accounts must contain the following information.
 - **Username:** A username is required to create an account.
 - **Email:** A unique email is required to create an account.
 - **Password:** A password that will be hashed for security purposes is required.
 - **Accepted Terms of Use:** Must accept to create an account.
 - **Name:** A registered user must have a first and last name.
 - **Gender:** A registered user must specify a gender.
 - **Age:** A registered user must have an age.
 - **Photo:** A profile shall have a picture of the registered user.
 - Has the ability to log into the service.
 - **Email:** Needs to login with a unique email.
 - **Username:** Needs to login with a unique username.
 - **Password:** Needs to provide a password.
 - Has the ability to communicate with other users.
 - Has the ability to create a student account.
 - Has the ability to create a landlord account
 - **Student:** A user who signed up with a .edu email is denoted a student.
 - Has the ability to create a group.
 - Has the ability to invite other student users to a group.
 - Has the ability to invite landlords to groups.
 - **Profile:** A profile will display information / attributes of a student user.
 - **Institution :** An institution will be part of a profile
 - **Major:** A major will be part of a student profile
 - **Schedule:** Needs to specify if they have a night or day schedule
 - **Hobby:** A hobby will be part of a student profile.

- **Personalities:** A personality will be part of a student profile.
- **Social Media:** A link to social media outlets.
- **Landlord:** A user who signed up with a non .edu email is denoted as a landlord.
 - Has the ability to post new listings.
 - Has the ability to create leases.
 - Has the ability to add students to a lease.
 - **Profile:** A profile will display information / attributes of a landlord.
 - **Listings:** The listings that they have posted
 - **Rating:** A list of ratings given by students.
 - **Student Ratings:** Number of students that have rated the landlord.
- **Admin User:** User that has the ability to moderate the service.
 - Can validate listings and make them public on the platform.
 - Can remove listings from the platform.
 - Can answer reports made by other users on the platform.
 - **Account Contains:** All registered user accounts must contain the following information.
 - **Username:** A username is required for all admin users.
 - **Email:** A unique email is required for all admin users.
 - **Password:** A password that will be hashed is required for all admin users.
- **Listing:** A representation of a housing unit that was posted by a landlord.
 - **Location:** Represents the physical location of a listing.
 - Represented by latitude and longitude coordinates.
 - **Photos:** Represents visual representations of a listing.
 - **Lease:** Represents an agreement between one or more students and a landlord which signifies that a lease between the parties exists.
 - Landlords can create leases for their listings.
 - Students can sign leases.

- **Amenities:** Represents the key features of a listing that are included.
 - Washer and Dryer
 - Wifi
 - Closets
 - Bathrooms
 - Furnished
 - Whiteboard
 - Living room
 - Kitchen
 - Patio
 - Parking
- **Favorite:** Specific listings that a user is interested in and wants to bookmark.
- **Match:** A representation of two student users who may be compatible roommates.
- **Groups:** A representation of a group of students that are matched and looking for listings together.
 - **Chat:** A way groups can communicate with each other on our platform.
 - Student users can share listing links in chat.
 - Student users can add a landlord user to a chat.
 - Landlord users can create a lease in a chat and invite all student users to sign the lease.

Prioritized Functional Requirements

1. Priority 1 - Required

1.1. Unregistered User

- 1.1.1. An unregistered user can create a new student or landlord account.
- 1.1.2. An unregistered user can view listings near an institution of their choice.
- 1.1.3. An unregistered user can view the Home page.
- 1.1.4. An unregistered user can view the Terms of Service page.
- 1.1.5. An unregistered user can view the FAQ page.
- 1.1.6. An unregistered user can view the Features pages.
- 1.1.7. An unregistered user can view the About page.

1.2. Registered User

- 1.2.1. A registered user should be able to login to with their username and password.
- 1.2.2. A registered user should be able to logout of their account.
- 1.2.3. A registered user should be able to submit a forgotten password form.
- 1.2.4. A registered user should be able to submit a forgotten email form.
- 1.2.5. A registered user should be able to change their username.
- 1.2.6. A registered user should be able to change their email address.
- 1.2.7. A registered user should be able to change their password.

1.3. Students

- 1.3.1. A student user must have a verified edu email.
- 1.3.2. A student user must have a completed profile.
- 1.3.3. A student user shall be able to view the student dashboard.
- 1.3.4. A student user shall be able to view student user profiles.
- 1.3.5. A student user shall be able to message another user.
- 1.3.6. A student user should be able to add other student users to a group.
- 1.3.7. A student user should be able to add a landlord to a group.
- 1.3.8. A student user shall be able to search for listings.
- 1.3.9. A student user shall be able to edit their personality.
- 1.3.10. A student user should be able to edit their schedule.
- 1.3.11. A student user should be able to edit their hobbies.
- 1.3.12. A student user shall be able to filter listings by price.
- 1.3.13. A student user shall be able to filter listings by amenities.
- 1.3.14. A student user shall be able to filter listings by distance from university.
- 1.3.15. A student user shall be able to filter roommate selections by personality.
- 1.3.16. A student user shall be able to filter roommate selections by major.

- 1.3.17. A student user shall be able to filter roommate selections by hobbies.
- 1.3.18. A student user shall be able to filter roommate selections by schedule.
- 1.3.19. A student user should be able to view the location of a listing on a map.
- 1.3.20. A student user shall be able to favorite a listing.

1.4. Landlords

- 1.4.1. A landlord user shall be able to view the landlord dashboard.
- 1.4.2. A landlord user shall be able to view their own profile.
- 1.4.3. A landlord user shall be able to post listings.
- 1.4.4. A landlord user shall be able to edit their listings.
- 1.4.5. A landlord user shall be able to view student profiles.
- 1.4.6. A landlord user will be able to respond to student user messages.
- 1.4.7. A landlord user shall be able to delete their listings.

1.5. Admin

- 1.5.1. Admin users should be able to approve or deny listings created by landlords.

2. Priority 2 - Desired

2.1. Unregistered User

2.2. Registered User

- 2.2.1. A registered user should be able to delete a chat.

2.3. Students

- 2.3.1. A student user should be able to change their university.
- 2.3.2. A student user shall be able to rate landlords.

2.4. Landlords

- 2.4.1. A landlord user shall be able view historical listing data.
- 2.4.2. A landlord user shall be able to repost their listing once expired.

2.5. Admin

- 2.5.1. Admin users should be able to edit registered user profiles.

3. Priority 3 - Opportunistic

3.1. Unregistered User

3.2. Registered User

3.3. Students

3.3.1. A student user shall be able to report a listing.

3.3.2. A student user shall be able to report another user.

3.4. Landlords

3.4.1. A landlord shall be able to report a listing.

3.4.2. A landlord shall be able to report another user.

3.5. Admin

3.5.1. Admin users should be able to disable non-admin user accounts.

3.5.2. Admin users should be able to remove a listing.

3.5.3. Admin users should be able to view reports.

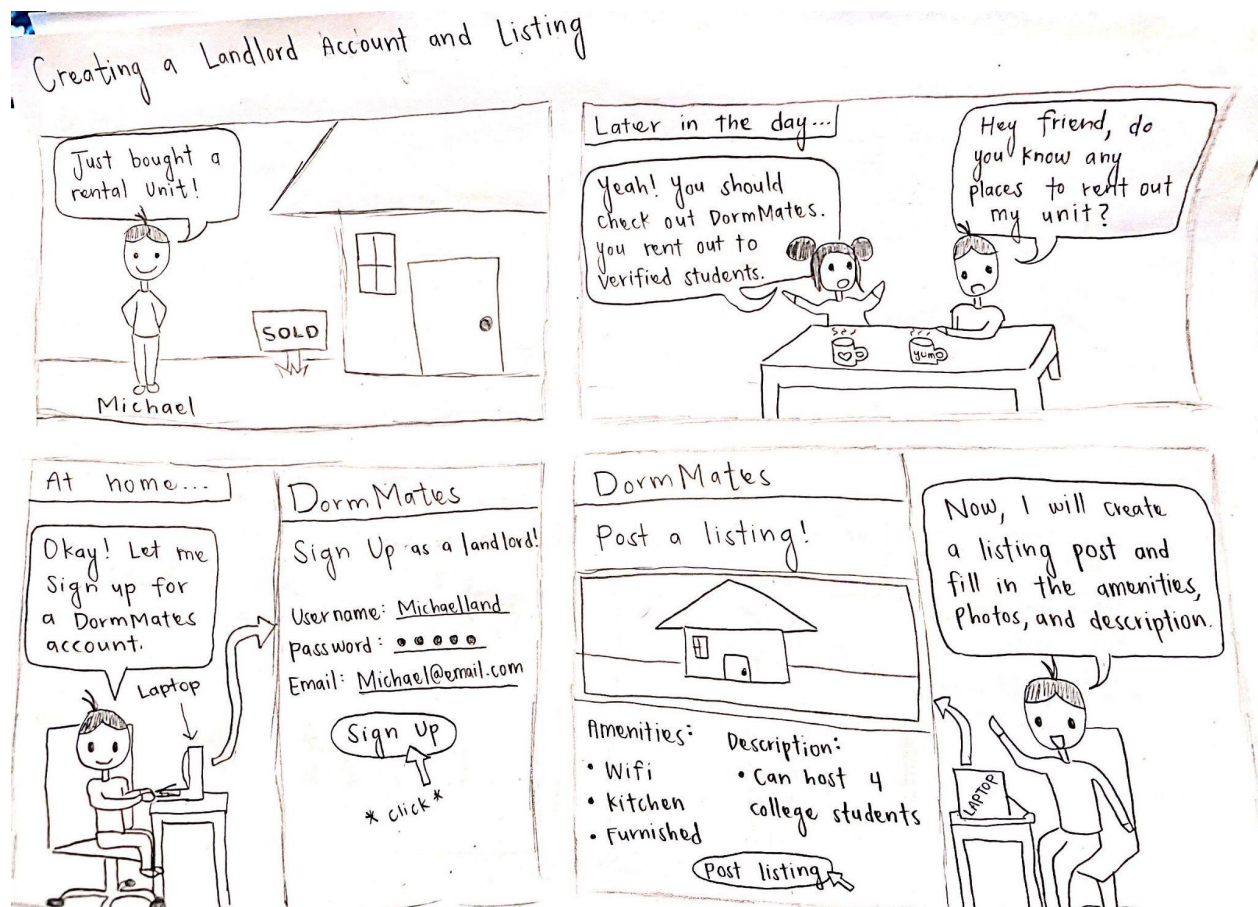
3.5.4. Admin users should be able to respond to reports.

UI Mockups and Storyboards

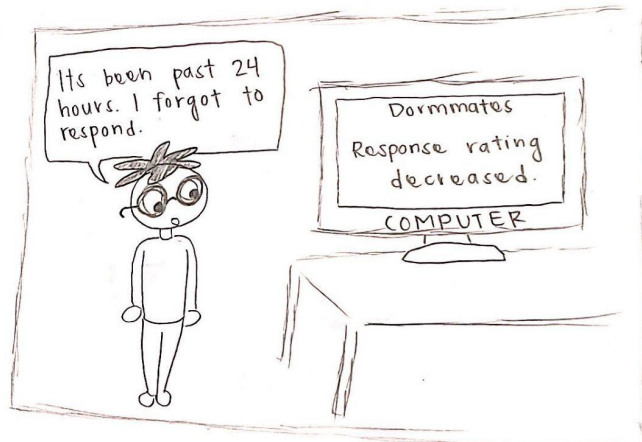
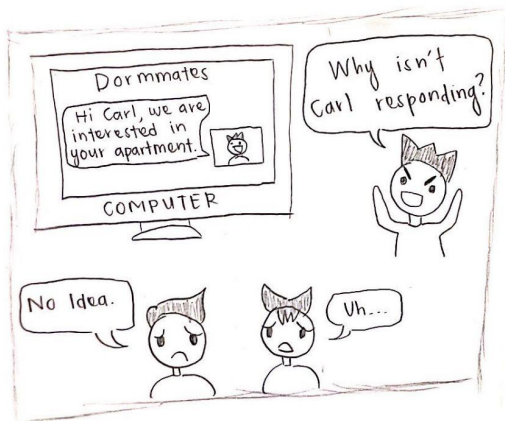
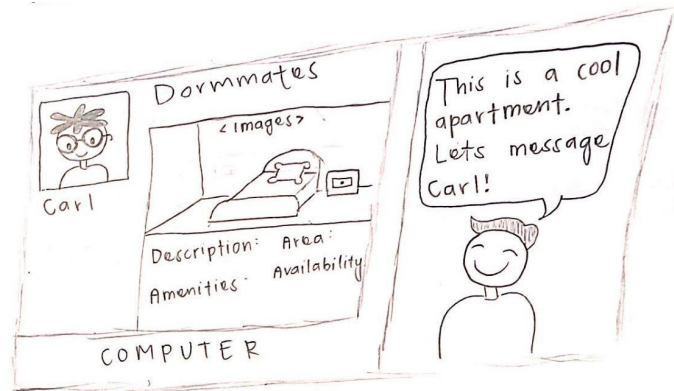
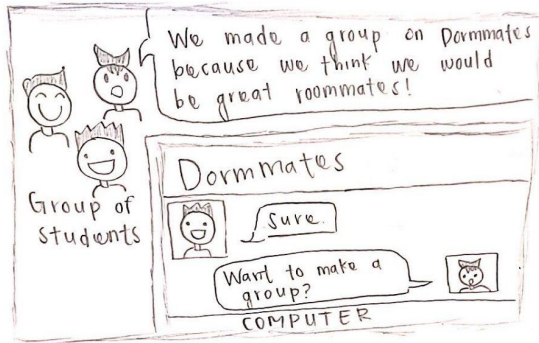
UI Mockups

[DormMatesWireframe](#)

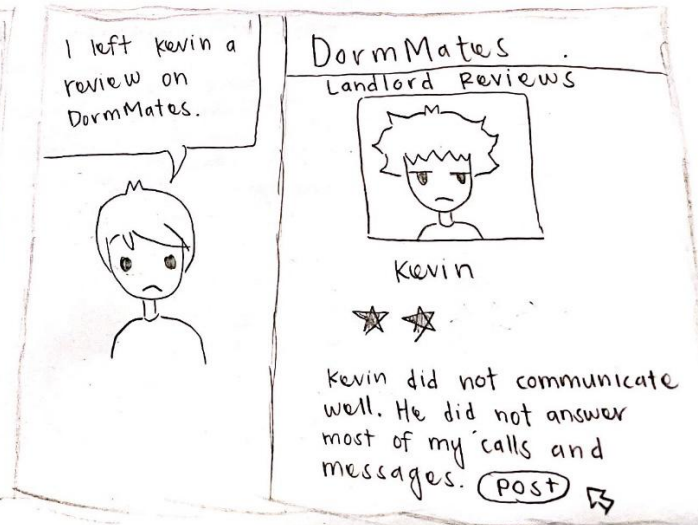
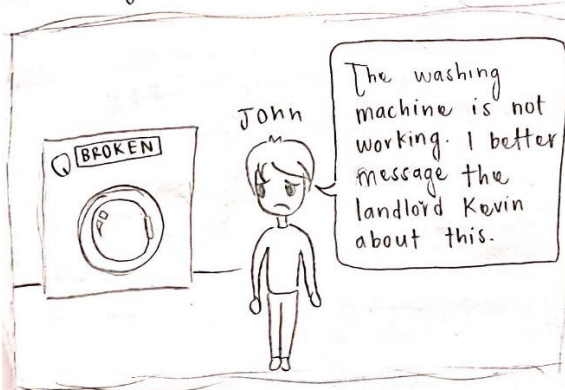
Storyboards



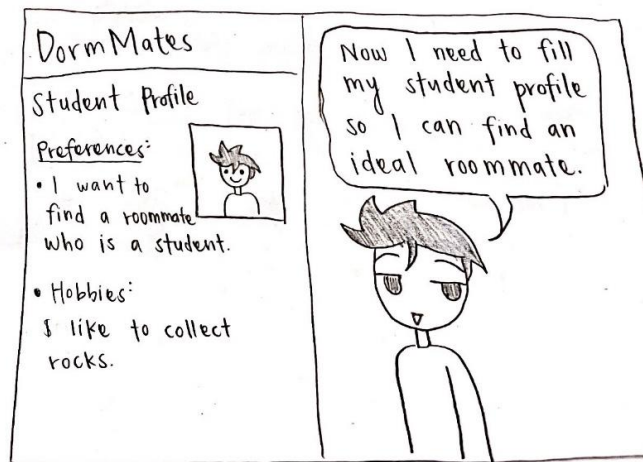
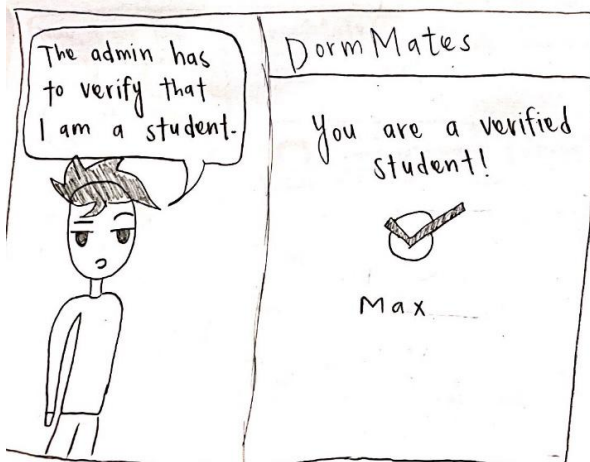
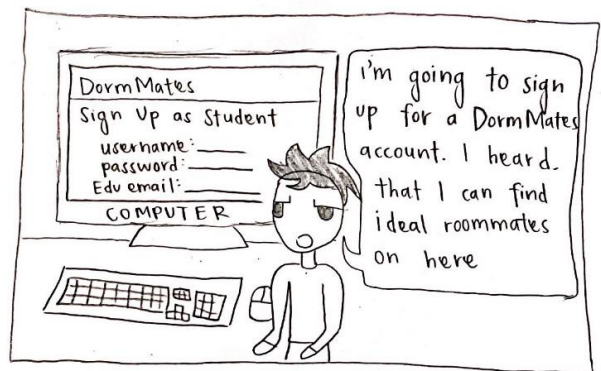
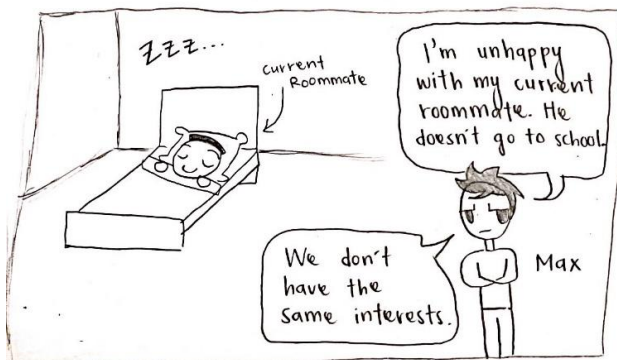
Landlord User Fails to Respond



Creating a Review




Student Creates a New Account



Student Needs Roommates

Time to create a DormMates account to find Roommates


Jose


DormMates

Sign Up as a Student!

Username: CoolGuy10
Password: •••••
Edu email: StudentJose@Harvard.edu

COMPUTER

Time to fill in my preferences and profile.




DormMates


Student Profile

Preferences

- Looking for 3 other students
- School: Harvard
- Schedule: M-F classes
- Major: Biology
- Budget: \$1000.00





JOSE

Now, I will look through my matches and message those who catch my eye.




DormMates

Student Matches


	Lily • Harvard • Biology	Budget: \$1000	<input type="button" value="message"/>
	Harry • Harvard • Biology	Budget: \$1000	<input type="button" value="message"/>
	Larry • Harvard • Biology	Budget: \$1000	<input type="button" value="message"/>


I messaged them! Now we can look through listings together through chat.





DormMates


Chat: Lily, Harry, Larry

 Hey guys, lets be roommates!

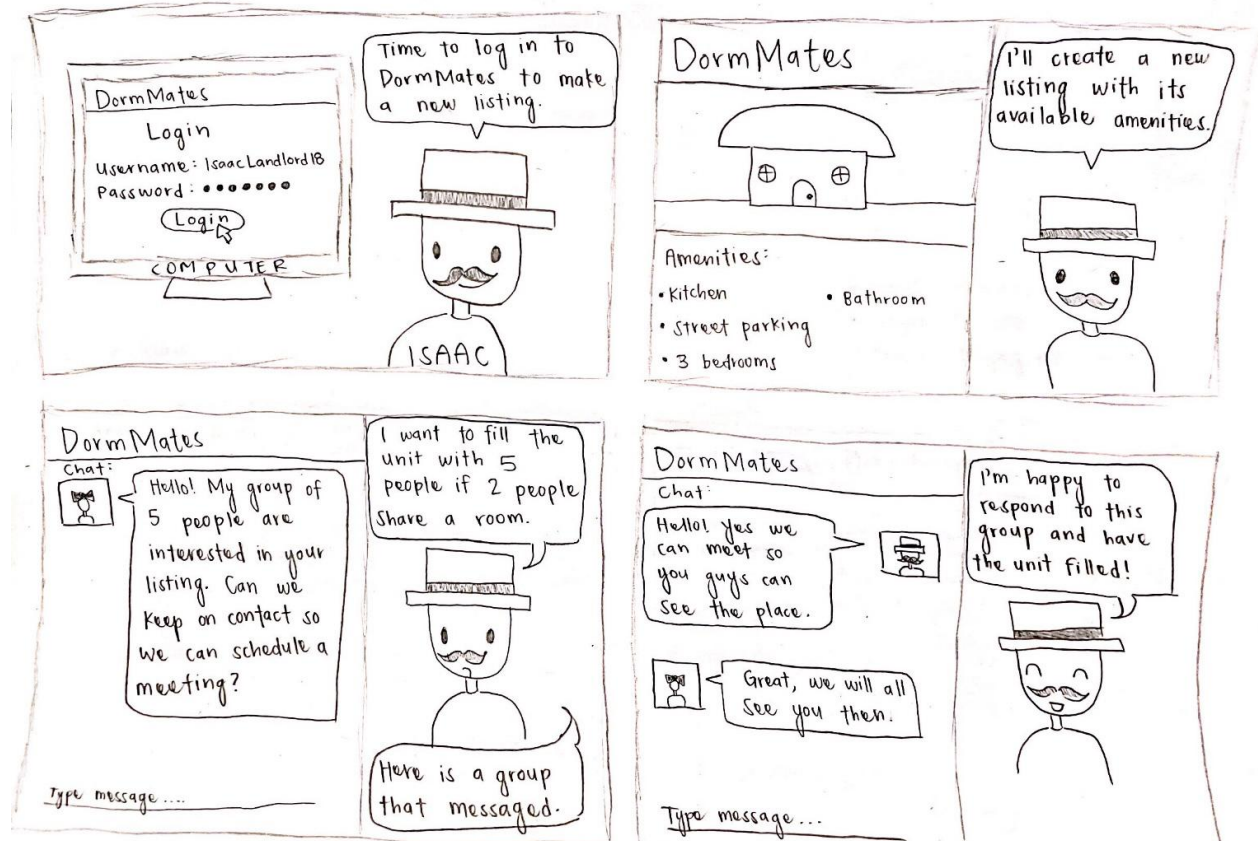
 Yes! Lets look for housing!

 Okay! I found some within our budget.

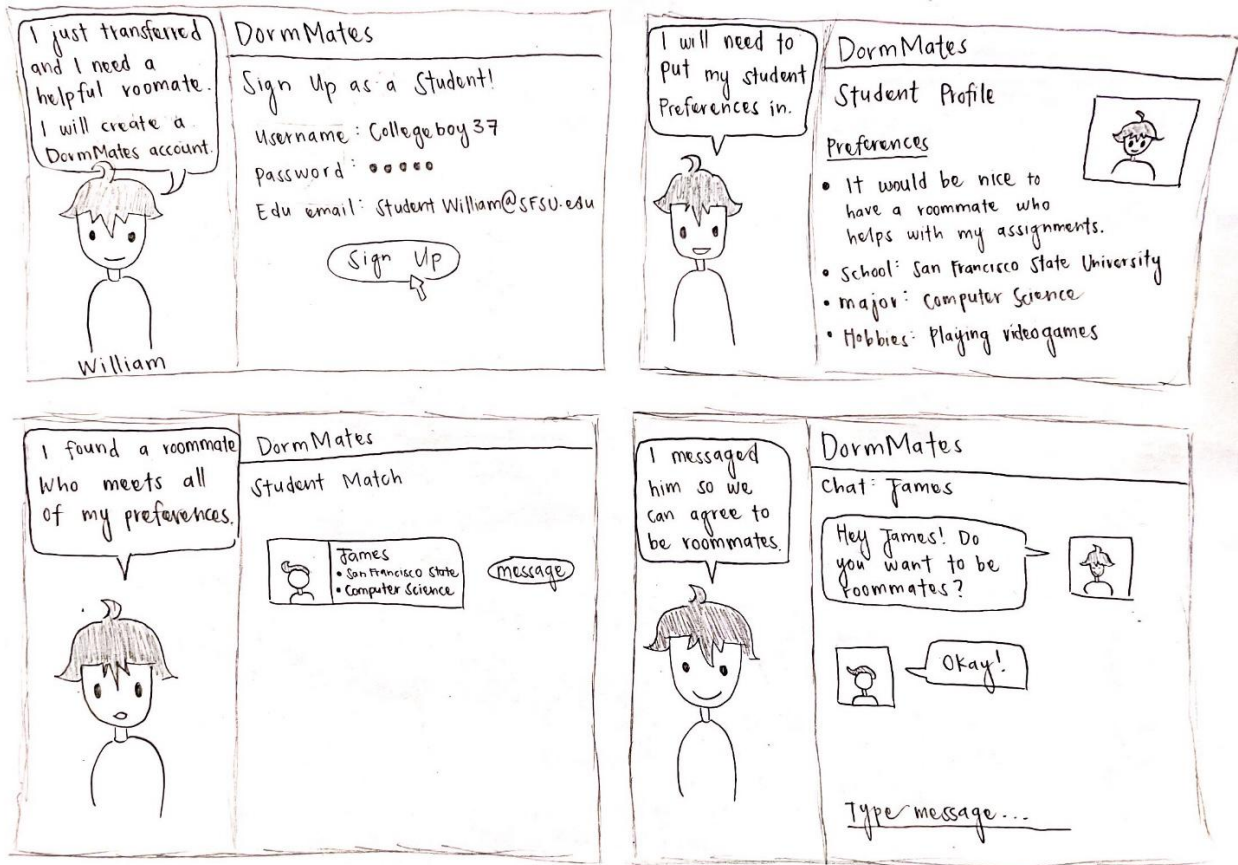
 Send links to them! Can't wait to be roommates!

 Cool! I look forward to it!

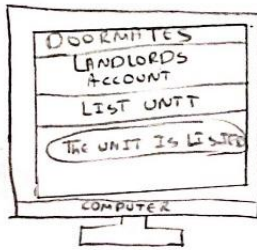
Landlord Wants to Fill Unit to Max Capacity



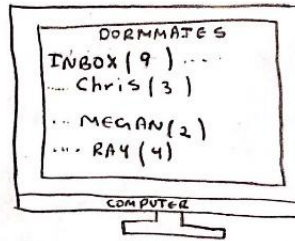
Student Rents a Property and Needs Help



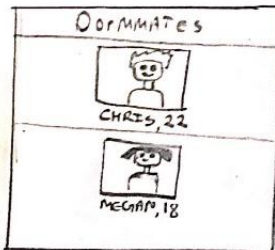
LANDLORD RESPONDING



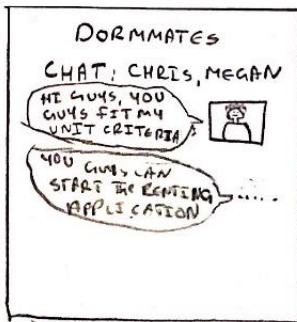
JUST LISTED
MY UNIT ON
DORMMATES!



I GOT MESSAGES
FROM STUDENTS!



I THINK BOTH
STUDENTS MEET
MY CRITERIA!



LET ME MESSAGE
THESE STUDENTS!



LANDLORD MANAGING MULTIPLE UNITS



LET ME CREATE LISTINGS ON DORMMATES



DORMMATES
LANDLORD ACCOUNT



MEGAN, HS

YOU HAVE CREATED
THE LISTINGS!

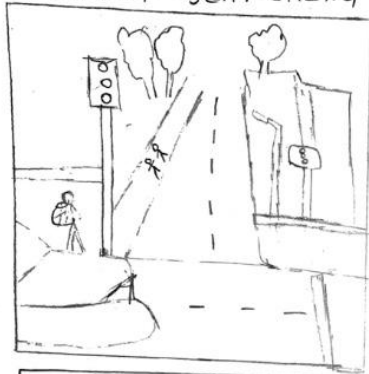
NICE! I CREATED
MULTIPLE LISTINGS!



STUDENT CHANGING THEIR LOCATION



STUDENT SEARCHING FOR ROOM



DORMMATES

SIGN UP

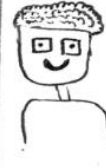
EMAIL

USERNAME

PASSWORD

SIGN UP

JUST
SIGNED
UP



DORMMATES

MENU

ZIP CODE PRICE FILTER

STONERIDGE \$1000

VINTAGE

MAP

DORMMATES

CHATS

HI, DO YOU WANT TO GO TO UCSB? 10:00 AM

HELLO, YES! 10:20 AM

NICE TO MEET YOU! 10:30 AM

NICE
PLATFORM



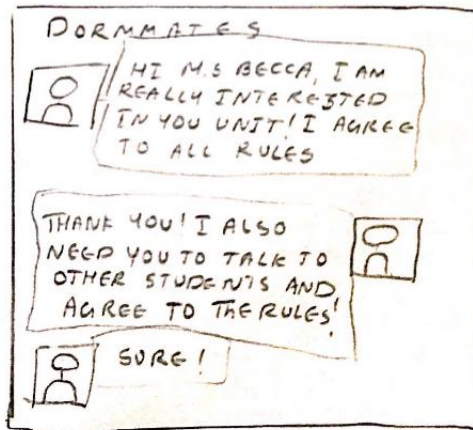
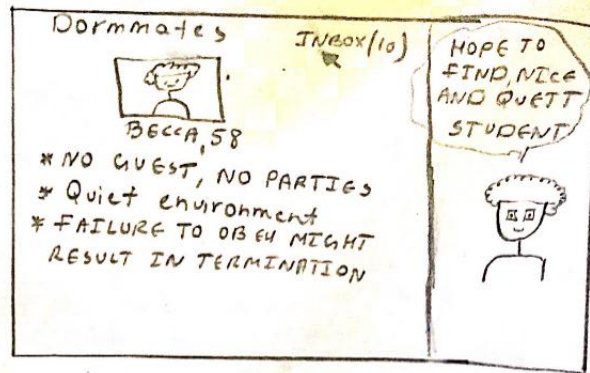
LANDLORD POSTING A LISTING AND UPDATING CAPACITY.



STUDENTS USING FAVORITES AND CONNECTING LANDLORDS



LANDLORD SETS LISTING RULES



High Level Database Architecture and Organization

Business Rules

- **Registered User:**
 - A registered user shall be able to log into one account.
 - A registered user shall have zero to many messages.
 - A registered user shall have zero to many sessions.
 - A registered user shall have zero to many reports.
- **Landlords:**
 - A landlord shall have zero to many ratings.
 - A landlord shall have zero to many listings.
 - A landlord shall have zero to many groups.
- **Student:**
 - A student shall have zero to many favorites.
 - A student shall have zero to many groups.
 - A student shall have one institution.
- **Messages:**
 - A message must have one registered user
 - A message must have one group.
- **Groups:**
 - A group shall contain two or more students
 - A group shall contain zero to one landlord
- **Listing:**
 - A listing shall have one landlord.
 - A listing shall have zero to many leases.
 - A listing shall have zero to many photos.
 - A listing shall have one amenities.
 - A listing shall have zero to many leases.
- **Photos:**
 - A photo shall have one listing.
- **Leases:**
 - A lease shall have one listing.
 - A lease shall have one student.
- **Favorites:**
 - A favorite shall have one listing.
 - A favorite shall have one student.
- **Reports:**
 - A report shall have two registered users.

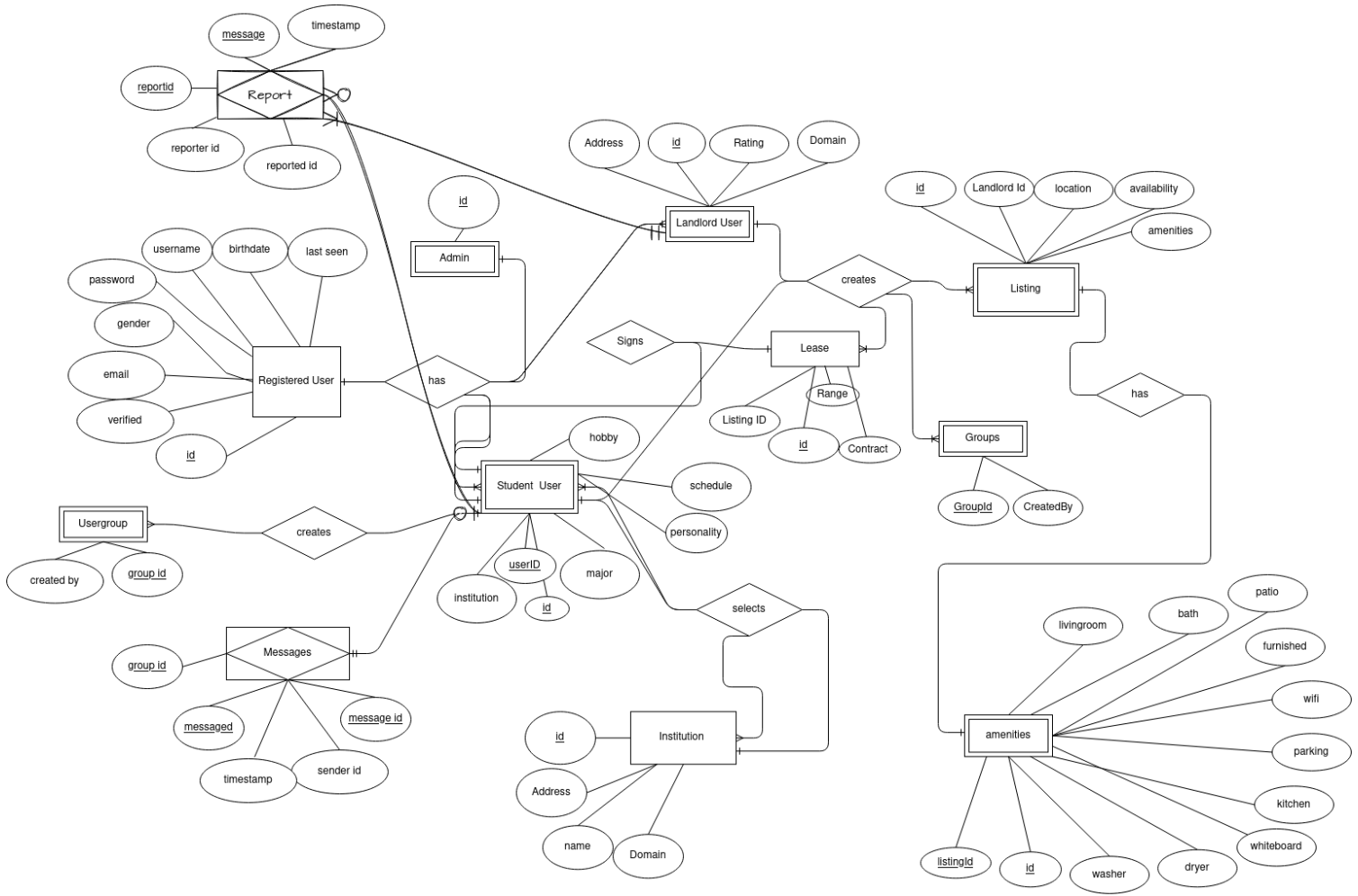
Entities

- **Registered User (Strong)**
 - registerd_id: key, alphanumeric
 - EmailAddress: alphanumeric
 - Password: key, numeric
 - Username: alphanumeric
 - Name: multivalued, alphanumeric
 - Birthdate: numeric
 - Gender: alphanumeric
 - Photo: Image BLOB
 - Last seen: numeric
 - Verified: boolean
- **Admin (Weak)**
 - id: key, alphanumeric
- **Student user (Weak)**
 - Id: key, alphanumeric
 - UserID: key, alphanumeric
 - Major: alphanumeric
 - Personality: alphanumeric
 - Schedule: alphanumeric
 - Institution: alphanumeric
 - Hobby1: alphanumeric
 - Hobby2: alphanumeric
- **Landlord user (Weak)**
 - Id: key, alphanumeric
 - Rating: numeric
 - UserID: weak key, numeric
 - NumOfRatings: numeric
- **Institution (Strong)**
 - Id: key , numeric
 - Name: alphanumeric
 - Domain: alphanumeric
 - Address: alphanumeric

- **Listing (Weak)**
 - Id: key, alphanumeric
 - LandlordID: weak key, alphanumeric,
 - Location: alphanumeric
 - Description: alphanumeric
 - Availability: alphanumeric
 - Amenities: alphanumeric
- **Amenities (Weak)**
 - Id: key, alphanumeric
 - ListingId: weak key, alphanumeric
 - Description: alphanumeric
- **Report (Weak)**
 - ReportId: key, alphanumeric
 - ReporterId: weak key, alphanumeric
 - ReportedId: weak key, alphanumeric
 - Message: alphanumeric
 - Timestamp: numeric
- **Favorite (Weak)**
 - ListingId: key, alphanumeric
 - UserId: weak key, alphanumeric
- **Lease (strong)**
 - LeaseId: key, alphanumeric
 - ListingId: weak key, alphanumeric
 - Contract: alphanumeric
 - Range: composite, numeric
- **Groups (Strong)**
 - GroupId: key ,alphanumeric,
 - CreatedBy: weak key, alphanumeric
- **UserGroup (Weak)**
 - GroupId: key, alphanumeric
 - UserId: weak key, alphanumeric
- **Messages (Weak)**
 - MessageId: key, alphanumeric
 - GroupId: key, alphanumeric
 - SenderId: weak key, alphanumeric
 - Message: alphanumeric
 - Timestamp: numeric

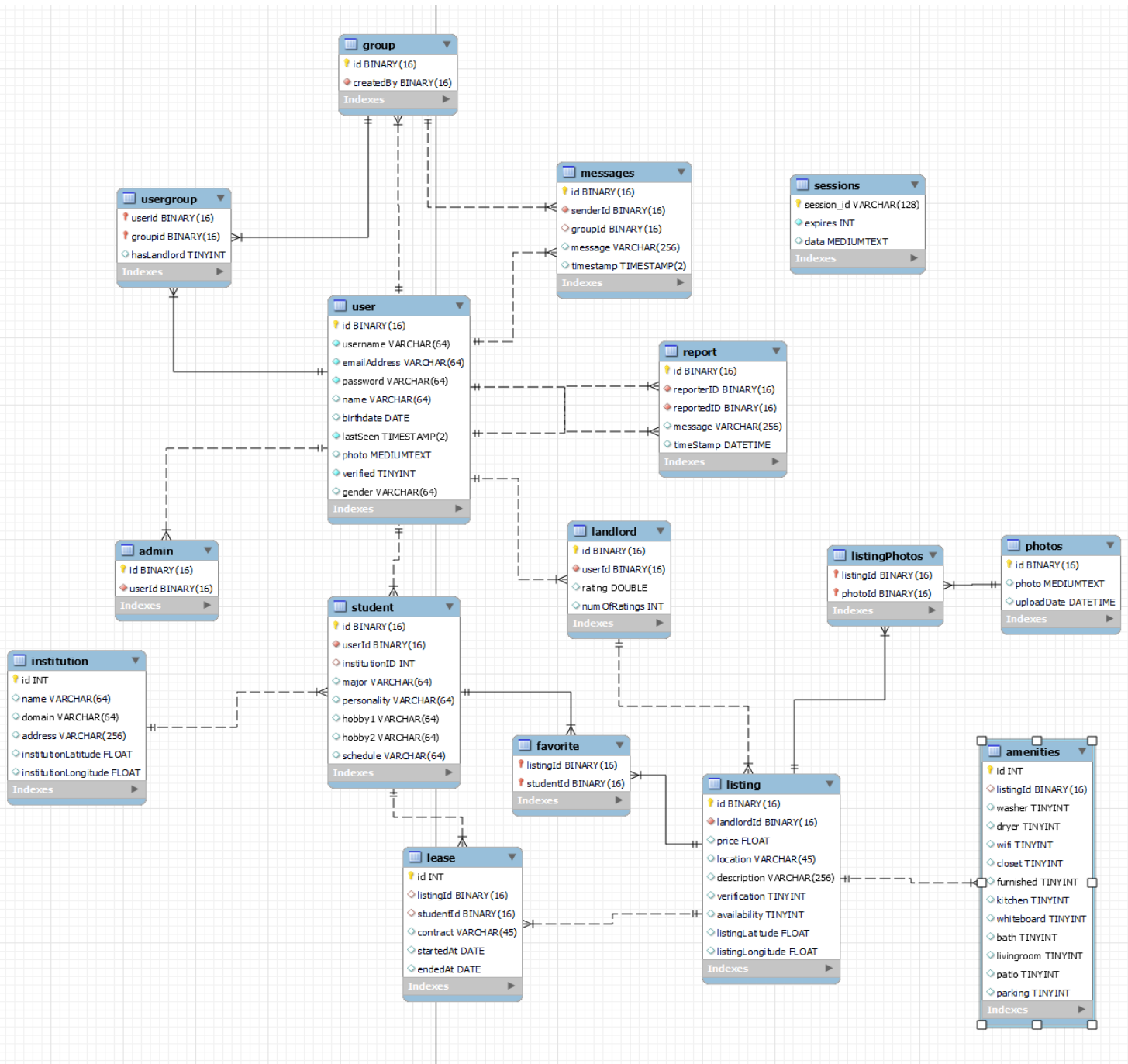
ERD Diagram

[Link to diagram \(draw.io\)](#)



Database Model

[DB SQL](#) (pastebin.com)



DBMS / Information

DBMS:

For the Database, we will be using MySQL because it is a relational database (requirement) and it is what we are most familiar with as a team.

Media Storage:

We will be storing all media in the database as BLOBs. We plan to use Multer to accomplish this.

Search Filer / Architecture and implementation:

For the search filters, we will be using SELECT statements with things like WHERE and LIKE clauses.

High Level APIs and Main Algorithms

- **Authentication API**

- **Registration**

- Post Request: When the backend receives a post request when a user signs up for an account using an email, username, and password. If all the inputs are valid then a successfully registered message is shown. Otherwise a responding failure to register account message is displayed.

- **Login**

- Post Request: The backend receives a post request when a user attempts to login using either a combination of their username and password or their email and password. The backend responds to the request by validating the login by checking if the user exists in the database, their email is correct, their username is correct, and/or their password is correct. When either the user doesn't exist or the login credentials are incorrect then the backend responds with an error message.

- **Chat API**

- **Send**

- Post Request: The backend receives a post request when a user begins a chat with one or more users. The backend checks for an id that matches with the group. If there are no groups that match an id then the chat is not sent.

- **Read**

- Get Request: When a user attempts to open a chat the backend receives a get request. The backend checks for an id that matches with the group and displays an array of data when found. If a group id is not found then an error is to be displayed.

- **Search API**

- **Filter Listing Search**

- **Get Request:** When a user views listing the backend receives the get request. The backend checks to see which listing id matches the user's requirements based on query parameters and displays all listings that meet the criteria. If no listing that meets the user's requirements is found then the backend responds with a message stating that zero listings were found that match the requirements.

- **Filter Roommate Search**

- **Get Request:** When a student user views the list of all available roommates near them the backend receives the get request. The backend searches for student ids that meet the student user's requirements based on query parameters and display roommate options that meet them. If no roommate option meets the user's preferences then the backend sends a message to be displayed stating no roommates meet the requirements.

- **Find Institutions**

- **Get Request:** When a user searches for institutions the backend receives a get request. The backend searches for attributes that are similar to the user's input and displays them. If no institutions are similar to the user's input then a message displays stating no institution can be found.

- **Rating API**

- **Landlord Rating**

- **Post Request:** A user can rate a landlord that they have rented from. The backend saves the input and notify the user that they have successfully rated the landlord. If the backend fails to do so the user receives a notification through an error message.
 - **Get Request:** When a user views a landlord, the user sees the rating of the landlord. The backend receives a get request with the landlord's id. If the backend fails to find an id then the backend responds with an error message.

- **Listing API**

- **Add a listing**

- **Post Request:** When a landlord user attempts to upload a listing the backend receives a post request. When all data fields are confirmed to be valid the backend saves the listing and displays that the listing is successfully added. If any of the data fields are invalid then the backend displays an error message stating that one or more fields are invalid.

- **Edit a listing**

- **Put Request:** A landlord user has the ability to edit listings they already uploaded. The backend receives the post request and updates the data fields that the landlord user has changed by first checking if the listing exists in the database by searching for the listing id.

- **Delete a listing**

- **Delete Request:** When a landlord user wants to delete one of the listings that they have on the service the backend receives a delete request. The backend searches for a matching listing id and removes the matching listing from the database. If no listing id is found then the backend displays an error message stating the listing could not be deleted.

- **Favorites API**

- **Favoriting a listing**

- **Post Request:** When a student favorites a listing the backend searches in the database for the listings id. If the backend finds the listing id it saves the listing id and pairs it with the student user id and displays a messaging stating that the listing was successfully favorited. Otherwise the backend displays a message stating that the listing was not able to be favorited.

- **Favorites List**

- **Get Request:** A student user viewing their favorites list sends a get request to the backend. The backend searches for listing ids and matches them with the student user's id and displays all listings that match. If no matches are made then the backend sends a message that displays no listings are favorited.

- **Lease API**

- **Create a lease**

- Post Request: A landlord adds a lease to a listing. The backend shall match the lease with the listing and save them in the database and display a message stating that the lease was successfully uploaded. If this fails then the backend sends a message stating that the lease was not uploaded.

- **Sign a lease**

- Post Request: A student user shall be able to sign a lease on listing. The backend then matches the lease with the student id and saves them within the database and displays a message stating that the lease was signed. If an error occurs then the backend sends a message stating that the lease was unable to be signed.

- **View a lease**

- Get Request: When a user views a listing a lease is displayed within that listing. The backend searches the database for a listing id to match with a lease id to display the lease.

- **Report API**

- **Create a report**

- Post Request: When a user reports a listing or another user the backend receives a post request. The backend searches for the id of either the listing or user being reported. If found then the backend sends a message stating the report was received. Otherwise the backend sends an error message.

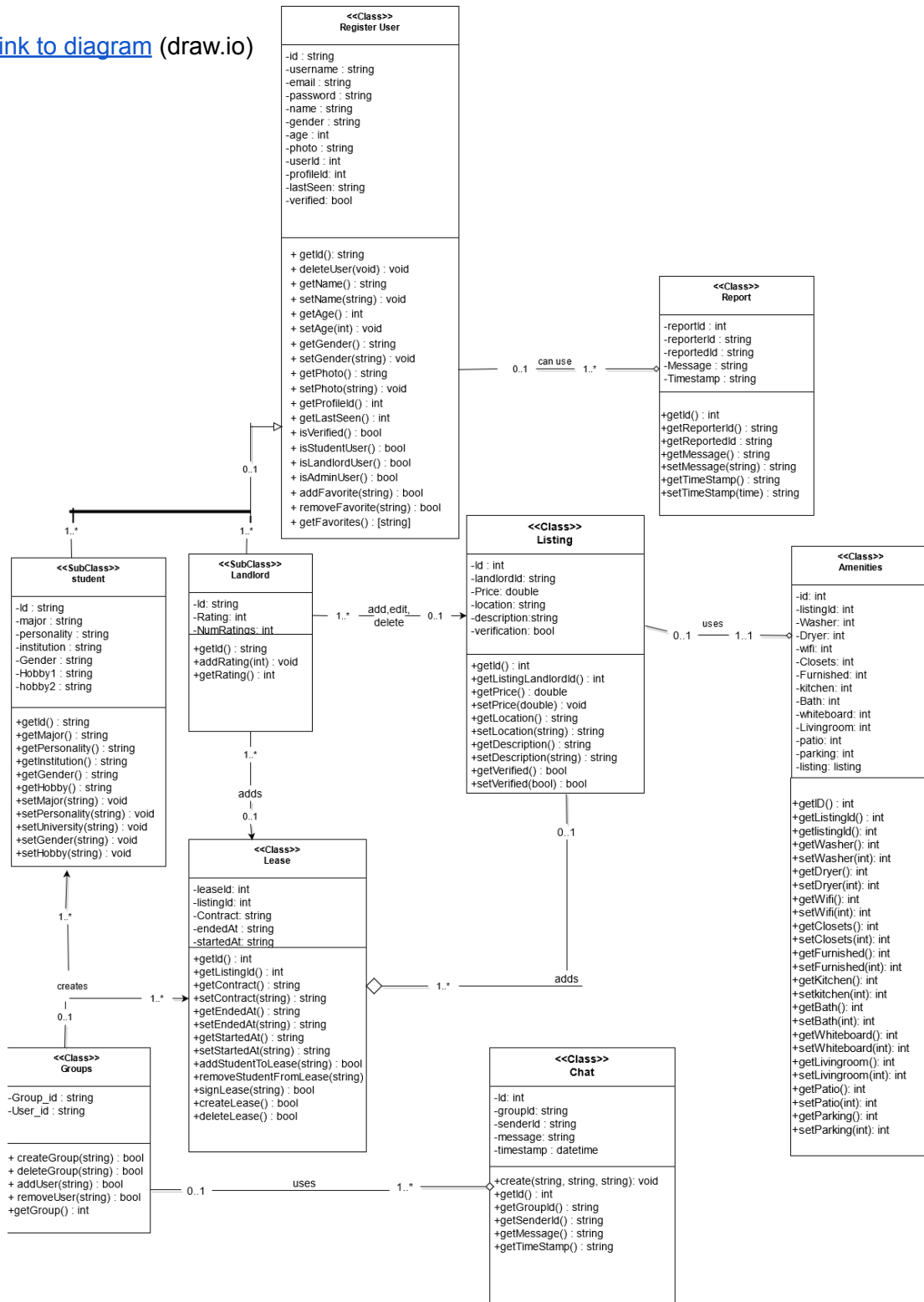
- **Rating Algorithm**

- Our service shall implement a rating algorithm that for when users attempt to contact a landlord and the landlord fails to reach them after a certain amount of time passes then their ratings will drop.

No new software has been added to the list of softwares our team has decided on using.

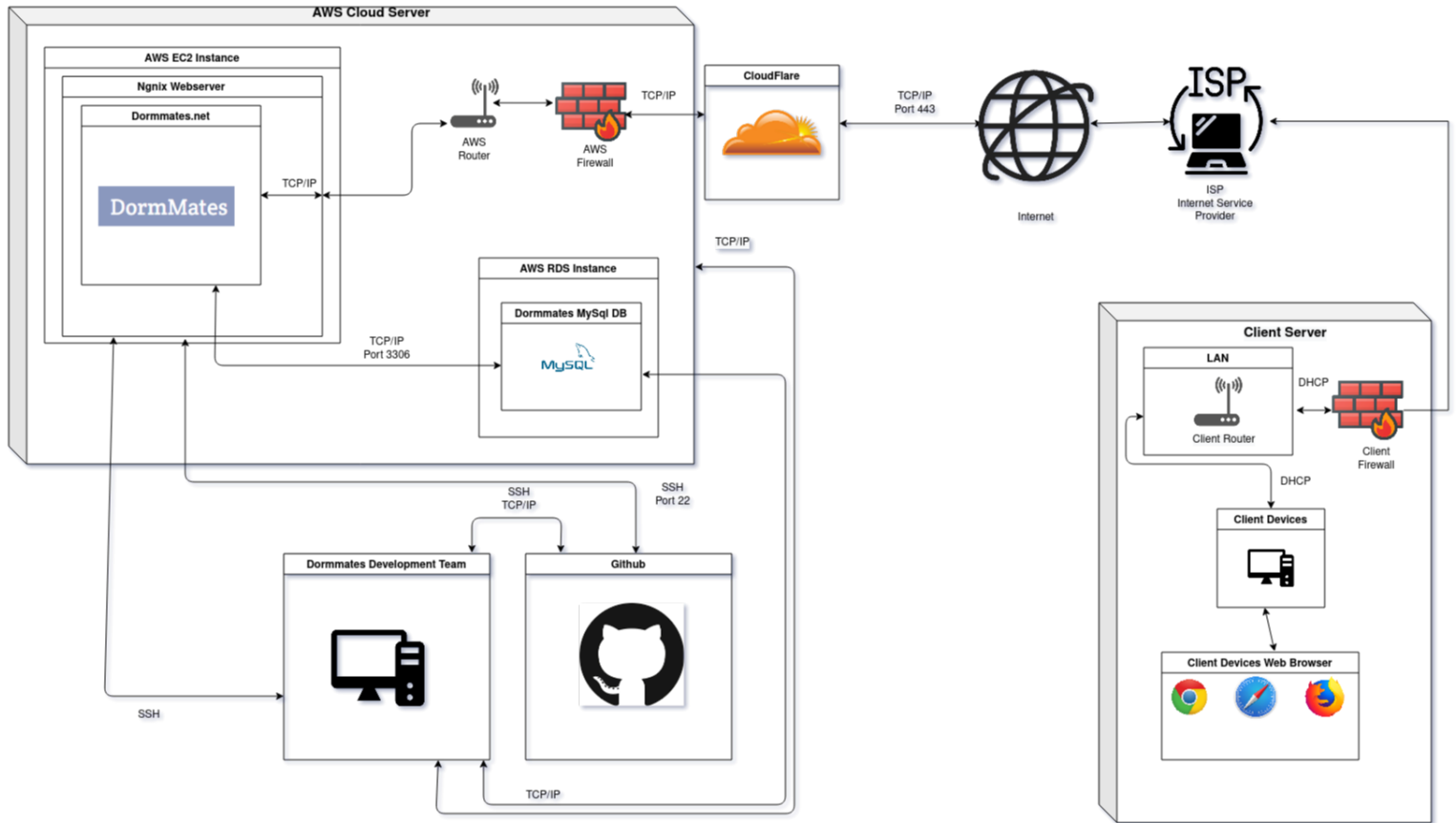
High Level UML Diagrams

[Direct link to diagram \(draw.io\)](#)

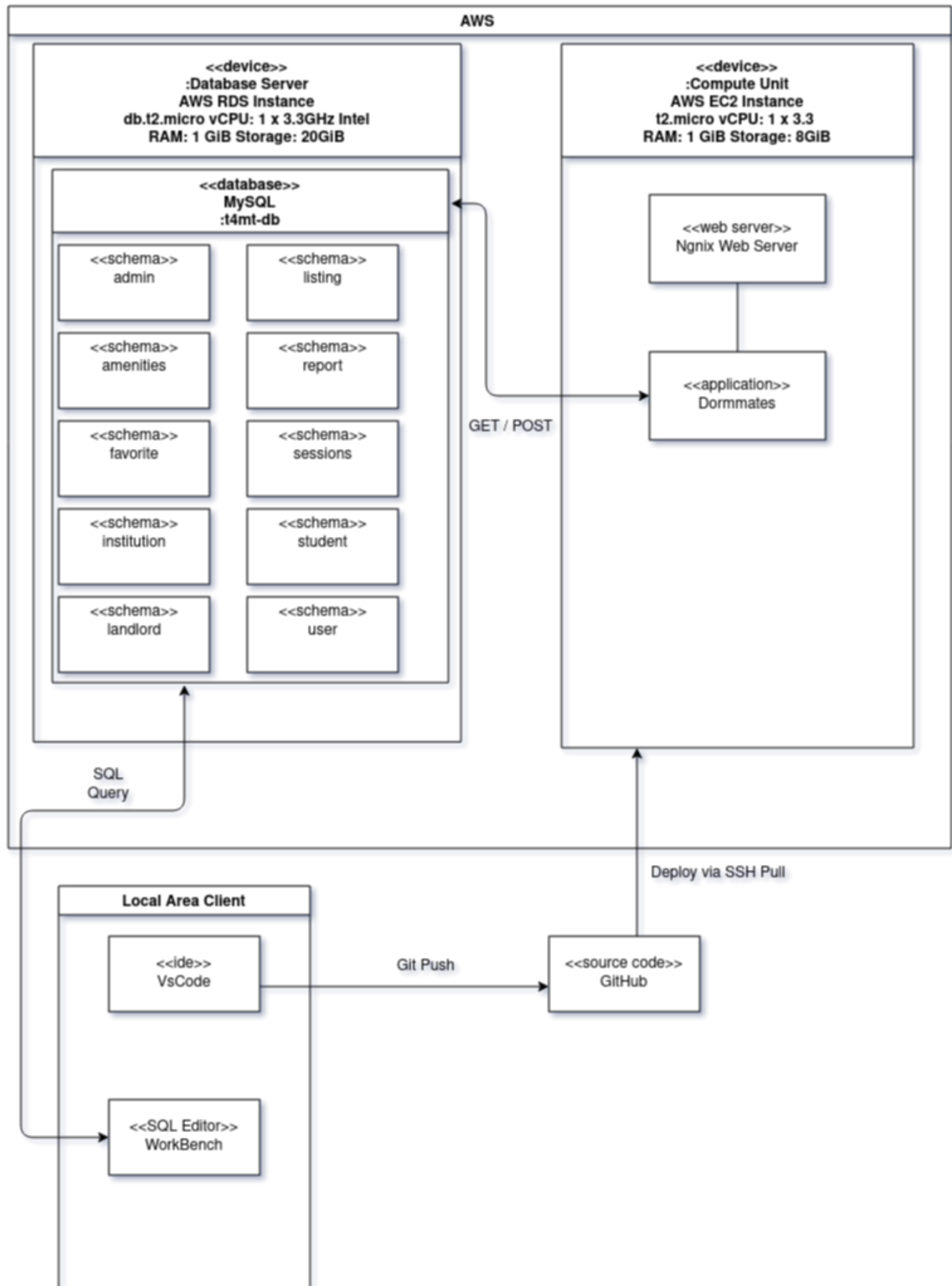


High Level Application Network and Deployment Diagrams

Network Diagram



Deployment Diagram



Key Risks

Skills

- **Writing**
 - Brainstorming the ideas in a document.
 - Needing more ideas to write in a document.
 - Don't be afraid to write anything; mistakes are encouraged and we should address them as a team.
 - Asking questions about things that may not be clear.
- **Gathering information**
 - Refer to the professor's milestone documents.
 - Asking the discord to clarify if you are unsure about anything.
- **Professor expectations:**
 - The team will communicate with the professor via Discord and email to clarify on expectations.

Schedule

- **Do the work before the agenda's due time.**
 - Individuals should not wait until the last second to complete agenda tasks.
 - When the agenda is posted, individuals should mark it seen with the SWE stickers
 - When the prior tasks are done in the agenda, the individuals should mark it seen with the 100% sticker.
- **Accommodating for every team members' schedule (classes, personal)**
 - The team lead will be more aware of everyone's schedules and conduct meetings with them in mind.
- **Keeping track of tasks on Jira:**
 - The team lead will monitor Jira frequently and check on progress of tasks.
- **Timely meetings:**
 - Start the meeting within 5 minutes of the start time, end within 15 minutes of estimated time
- **Keep meetings productive:**
 - The agenda will include an estimated timeline for the meeting along with the listed tasks.
 - Agenda should include a max allocated time for each meeting.
 - The team lead should give out breaks during the meeting.
 - The team lead should ensure that all meetings are on task.

Technical

- **Organization:**
 - Milestone editor will be responsible for formatting all content.
 - The github master will ensure that all coding guidelines are being followed.
- **Using technologies.**
 - Making sure everyone is comfortable using Mysql
 - The backend team will individually review/refresh mysql before starting the first task.
 - Making sure everyone understands the framework for frontend.
 - Making sure all backend teammates are comfortable using javascript.
 - Making sure all backend teammates are comfortable with Express.
 - Making sure everyone knows how to use the discord bot.
 - Make an effort to read any documentation for said technology.

- **Knowledge of frameworks**
 - Reading documentation for frameworks
 - Looking at examples of frameworks being used
 - Asking for help or for clarity

Teamwork

- **Making sure that everyone is contributing to the project:**
 - Alerting the team lead if someone isn't contributing as much as they should.
 - The team lead will be more aware of individual contributions and address the lack of contributions swiftly.
 - Make sure every team member is involved and happy with their contributions
- **All members are attending team meetings:**
 - If they can't attend meetings, they are watching the meeting recordings.
 - Any team member who must watch a recorded meeting should create a writeup commenting on what they missed to show that they are on the same page.
 - The team lead will send out reminders for all meetings.
- **Productivity:**
 - The team lead will create internal deadlines and ensure that they are being met.
 - Ensuring deadlines and requirements are made clear
- **Ensuring all members agree on deadlines**
 - Clear and concise tasks on Jira with internal deadlines.
 - Ensuring that the majority decision policy is always in effect.
- **All members completing their task on time:**
 - The team lead will monitor tasks on Jira to ensure progress is being made.
 - Team members will give notice of any issues they are having or complications.

Legal/Content

- **Copyrighted Content**
 - All teammates must verify any content used on the site is not copyrighted
 - Ensuring we give credit to artwork.
 - Ensuring that all software used in our project is open source/has a license that allows us to use it freely.
- **Ensuring all users accept and respect our terms of use:**
 - Users will not be able to register without accepting the terms and use
- **Ensuring users should be 18 and over:**
 - We'll require all unregistered users to confirm that they are 18 or older when creating a new account.

Project Management

For project management we are using Jira by Atlassian.

In milestone 2 and all future milestones, the team lead works with the frontend and backend leads to dissect the milestone document and break down high level tasks into many sub tasks that are then assigned to individual members through Jira.

Each sub task has a due date, description, and child tasks. The due date is when the task and all of its child tasks are due, the description is a high level description of the task, and the child tasks are itemized tasks that break the sub task down into checkpoints that should be completed in the order they are listed.

Detailed List of Contributions

Andrei

- Managed the project by creating tasks on Jira and dispatching them to the team.
- Managed the project by keeping up with individual members of the team to track progress and give feedback.
- Met with the frontend and backend team frequently to give advice and feedback on work being done.
- Hosted Zoom meeting where the team worked on addressing Milestone 1 feedback.
- Hosted Zoom meetings where the team worked on Milestone 2 tasks.
- Hosted meetings over Discord with backend and frontend teams to brainstorm implementation.
- Hosted pair programming sessions with the team for setting up the project development environment.
- Hosted pair programming session with the backend team for creating the backend institution search endpoint.
- Hosted pair programming session with the frontend team for connecting the frontend and the backend.

Jonathan

- Attended all team meetings.
- Communicated with the team through Discord often.
- Hosted meetings over Discord with the backend team to brainstorm backend implementation.
- Hosted pair programming session for creating backend user model, controller, route, and endpoint.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Implemented the User API.
- Worked on the High Level Database Architecture and Organization task.
- Attended all backend pair programming sessions with the team.

Meeka

- Attended all team meetings.
- Communicated with the team through Discord often.
- Hosted meetings over Discord with the frontend team to brainstorm frontend implementation.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on the Use Case Mockup / Storyboard milestone task.

- Created Wireframes for the vertical prototype.
- Created Pug views for wireframes they created.
- Created CSS styles for the frontend.
- Attended all frontend pair programming sessions with the team.

Jimmy

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on documenting what API routes the backend team needed to implement.
- Worked on the High Level UML Diagrams task.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Attended all backend pair programming sessions with the team.

Alexandre

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on the High Level APIs and Main Algorithms task.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Created documentation for the backend code.
- Created documentation for the deployment process of the application.
- Attended all backend pair programming sessions with the team.

Sayed

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on the Use Case Mockup / Storyboard milestone task.
- Created Wireframes for the vertical prototype.
- Created Pug views for wireframes they created.
- Created CSS styles for the frontend.
- Attended all frontend pair programming sessions with the team.

Ahad

- Attended all team meetings.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Compiled a list of all colleges in Northern California and inserted them into our database.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Attended some backend pair programming sessions with the team.